Discrete Applied Mathematics 18 (1987) 211–225
North-Holland



# A PARALLEL BRANCH AND BOUND ALGORITHM FOR THE QUADRATIC ASSIGNMENT PROBLEM

Catherine ROUCAIROL

*MASI, Université Paris 6, 4 Place Jussieu, 75230 Paris, and INRIA, BP. 105, Domaine de Voluceau, Rocquencourt, 78153 Le Chesnay Cedex, France*

Received 6 March 1987

We propose a parallel branch and bound algorithm for the quadratic assignment problem; this algorithm has been implemented on an asynchronous multiprocessor machine with shared memory (the Cray X-MP). For problems with size $n \geq 10$, the improvement in using $n$ processors is very close to $n$, and moreover very good results are obtained for a classical example from the literature with size 12.

## 1. Introduction

Quadratic assignment problems (QAP) even of moderate size ($n = 10$) are very hard to solve. We proposed some years ago a method which produces optimal solutions to QAP's for sizes up to $n = 12$ and good solutions for $n \geq 15$. This method was based upon a reduction procedure which splits the objective function into a linear term and a reduced quadratic term (2.2). This reduction enables easy computation of lower and upper bounds for the cost of the optimal solution (2.2). The last step was followed by a branch and bound procedure (3.3), but as its computational requirements grow exponentially with the problem size $n$, problems of practical size ($n = 20$) cannot be solved exactly due to excessive running time and memory requirements.

Therefore, the idea of realizing a parallel implicit enumeration of the solutions of the problem in order to accelerate the search, has naturally emerged.

As we proposed a general method for designing a distributed branch and bound algorithm, well suited for asynchronous MIMD computers or computer networks, we then adapted it to shared memory multiprocessor machines [5].

The structure of the B&B algorithm is such that it increases the number of parts of the tree that can be carried out in parallel: a 'polytomic' branching rule is used; it generates several successors of a node of the search tree and strongly constrains subproblems (many solutions are excluded from each subproblem), and it allows us to keep only little information about each node of the tree [3]. From experimental results, we analyze the performance of the algorithm (5.5).

## 2. A branch and bound procedure for QAP: Shiva's method

Assignment problems with quadratic objective functions can be found in different fields such as: economics (plant location problem), electronics (blackboard wiring problem), computer aided design in architecture (layout of hospitals), ergonomics (development of new type/writer keyboards).

We give an example of a facility location problem. Let us suppose that $n$ plants are to be assigned to $n$ locations. Here, $x_{ik}$ equals 1 if facility $i$ is placed in location $k$, 0 otherwise, $f_{ij}$ is the flow of 'material' between the facility $i$ and the facility $j$, $d_{kl}$ is the distance between location $k$ and location $l$. Then we may define the problem as follows:

$$\begin{cases} x_{ik} = 0 \text{ or } 1, \\ \sum_{i=1}^{n} x_{ik} = 1, \quad k = 1, \ldots, n, \\ \sum_{k=1}^{n} x_{ik} = 1, \quad i = 1, \ldots, n, \\ \text{Min} \sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{k=1}^{n} \sum_{l=1}^{n} f_{ij} d_{kl} x_{ik} x_{jl}. \end{cases}$$

In short, we can consider the assignment of facilities to locations as a permutation $p$ of the set $N = \{1, 2, 3, \ldots, n\}$ by setting $p(i) = j$ if facility $i$ is assigned to location $j$.

Then the QAP consists of finding a permutation $p$ that minimizes:

$$Z(p) = \sum_{i=1}^{n} \sum_{j=1}^{n} f_{ij} d_{p(i)p(j)}.$$

QAP's belong to the class of NP hard problems. Even problems of moderate size are very difficult to solve. We proposed some years ago a method based upon a reduction process. This reduction enables us to compute bounds more easily in the tree search procedure which will be described later. We briefly review this method and its last development SHIVA (for details, see [16], [12]).

### 2.1. Reduction procedure

The reduction of a matrix $M = (m_{ij})$, $(i, j) \in N \times N$, with nonnegative elements consists of finding $2n$ numbers $\alpha_i$, $\beta_j$ so that:

$M' = (m'_{ij}) = m_{ij} - \alpha_i - \beta_j$ is a matrix with nonnegative elements which has at least one zero (element such as $m'_{ij} = 0$) in each row and each column. If we apply this transformation to both matrices $F$ and $D$, we obtain two reduced matrices $F'$ and $D'$ and a new quadratic assignment problem with the objective function $Z'(p)$:

$$Z'(p) = \sum_{i} \sum_{j} f'_{ij} d'_{p(i)p(j)}$$

where $F' = (f'_{ij})$ with $f'_{ij} = f_{ij} - \alpha_i - \beta_j$, $D' = (d'_{kl})$ with $d'_{kl} = d_{kl} - \alpha'_k - \beta'_l$.

It can be shown [16] that the following relation holds between the objective function of the original problem $Z$ and the reduced problem $Z'$:

$$Z(p) = Z'(p) + K(p) - g \tag{1}$$

where $g$ is a positive constant: $g = (\sum_i \alpha_i)(\sum_l \beta'_l) + (\sum_k \alpha'_k)(\sum_j \beta_j)$, $K(p) = \sum_i k_{ip(i)}$ is the objective function of a linear assignment problem with respect to the matrix $K = (k_{ik})$ where

$$k_{ik} = \alpha_i \sum_l d_{kl} + \beta_i \sum_l d_{lk} + \alpha'_k \sum_j f_{ij} + \beta'_k \sum_j f_{ji}$$

$$- (n-1)(\alpha_i \alpha'_k + \beta_i \beta'_k) + \alpha_i \beta'_k + \alpha'_k \beta_i,$$

$$k_{ik} \geq 0.$$

Let us illustrate this method with an example.

Facilities $F_1$, $F_2$, $F_3$, $F_4$ are to be assigned to locations $L_1$, $L_2$, $L_3$, $L_4$.

$F =$

| | $F_1$ | $F_2$ | $F_3$ | $F_4$ |
|---|---|---|---|---|
| $F_1$ | | 8 | 8 | 12 |
| $F_2$ | 8 | | 5 | 2 |
| $F_3$ | 8 | 5 | | 11 |
| $F_4$ | 12 | 2 | 11 | |

$D =$

| | $L_1$ | $L_2$ | $L_3$ | $L_4$ |
|---|---|---|---|---|
| $L_1$ | | 1 | 4 | 3 |
| $L_2$ | 1 | | 2 | 1 |
| $L_3$ | 4 | 2 | | 3 |
| $L_4$ | 3 | 1 | 3 | |

Two reduced matrices $F'$ and $D'$ are obtained here (method red1) by substracting from each row, then from each column its minimum element.

$F' =$

| | | | | $\alpha_i$ |
|---|---|---|---|---|
| | 0 | 0 | 2 | 8 |
| 0 | | 5 | 0 | 0 |
| 0 | 5 | | 9 | 0 |
| 2 | 0 | 9 | | 2 |
| $\beta_j$ 8 | 0 | 0 | 2 | |

$D' =$

| | | | | $\alpha'_i$ |
|---|---|---|---|---|
| | 0 | 1 | 1 | 1 |
| 0 | | 0 | 0 | 0 |
| 1 | 0 | | 0 | 2 |
| 1 | 0 | 0 | | 1 |
| $\beta'_j$ 1 | 0 | 2 | 1 | |

$K =$

| | $L_1$ | $L_2$ | $L_3$ | $L_4$ |
|---|---|---|---|---|
| $F_1$ | 152 | 64 | 192 | 136 |
| $F_2$ | 30 | 0 | 60 | 0 |
| $F_3$ | 48 | 0 | 96 | 48 |
| $F_4$ | 74 | 16 | 120 | 70 |

$g = 80$

$$Z(p) = \sum_i \sum_j f'_{ij} d'_{p(i)p(j)} + \sum_i k_{ip(i)} - g$$

Due to the linear term $K(p)$ one can easily compute a lower and an upper bound of the optimal value.

### 2.2. Bound calculation

Let $P$ be the set of all permutations of $(1, \ldots, n)$. Consider $Z(p) = Z'(p) + K(p) - g$ for $p \in P$ and let $P^*$ be the optimal permutation: $Z^* = Z(p^*) = \min_p Z(p)$.

*Lower bound*

We use the technique of Gilmore [7] to compute a bound of $Z'(p)$. Gilmore shows that $\min_p \sum_{i=1}^{n} a_i b_{p(i)}$ can be calculated by ordering the elements of $a_i$ increasingly and the elements of $b_j$ decreasingly ('ordered product' of vectors $a$ and $b$).

Let us call PO $(f'_{i\cdot}, d'_{k\cdot})$ the ordered product of row $i$ of $F'$ and column $p(i) = k$ of $D'$, i.e.,

$$\mathrm{PO}(f'_{i\cdot}, d'_{k\cdot}) = \sum_{r=1}^{n} f'_{ij_r} \cdot d'_{kl_r}$$

where $f'_{ij_1} \leq f'_{ij_2} \leq \cdots \leq f'_{ij_n}$ and $d'_{kl_1} \geq d'_{kl_2} \geq \cdots \geq d'_{kl_n}$.

Setting $\mathrm{PO}(p) = \sum_{i=1}^{n} \mathrm{PO}_{ip(i)}$, we obtain $Z'(p) \geq \mathrm{PO}(p)$ and $Z(p) \geq \mathrm{PO}(p) + K(p) - g$ for all $p \in P$.

Let $B(p) + K(p)$ and let $B$ be the matrix with elements $\mathrm{PO}_{ik} + K_{ik}$. Then we get for example

$B =$

| | $L_1$ | $L_2$ | $L_3$ | $L_4$ |
|---|---|---|---|---|
| $F_1$ | 0 | 0 | 0 | 0 |
| $F_2$ | 0 | 0 | 0 | 0 |
| $F_3$ | 5 | 0 | 0 | 0 |
| $F_4$ | 2 | 0 | 0 | 0 |

matrix PO

$+$

| | | | | |
|---|---|---|---|---|
| 152 | 64 | 192 | 136 | |
| 30 | 0 | 60 | 30 | |
| 48 | 0 | 96 | | |
| 74 | 16 | 120 | 70 | |

matrix $K$

$=$

| | | | |
|---|---|---|---|
| 152 | 64 | 192 | 136 |
| 30 | 0 | 60 | 30 |
| 53 | 0 | 96 | 48 |
| 76 | 16 | 120 | 70 |

A permutation $P^B$ may be found by solving a linear assignment problem with cost matrix $B$.

Let $B^*$ be the final reduced matrix of the Hungarian method, i.e., $B^* = (B_{ik} - u_i - v_k)$ where $u_i$ and $v_k$ are the optimal dual variables. Then one has

$$B(p) = B^*(p) + B(p^B) \quad \text{for all } p \in P.$$

Our example yields

$$P^B = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 3 & 1 & 4 \end{pmatrix}, \qquad B(p^B) = 247 \quad \text{and}$$

$B^* =$

| | $L_1$ | $L_2$ | $L_3$ | $L_4$ |
|---|---|---|---|---|
| $F_1$ | 29 | 0 | 39 | 18 |
| $F_2$ | 0 | 29 | 0 | 5 |
| $F_3$ | 0 | 6 | 13 | 0 |
| $F_4$ | 1 | 0 | 15 | 0 |

Therefore, we get the lower bound

$$\underline{Z} = B(p^B) - g$$

which works out to $\underline{Z} = 247 - 80 = 167$.

*Upper bound*

This permutation immediately gives us the upper bound $\bar{Z} = Z(p^B) \geq Z(p^*)$. In our case $\bar{Z} = 190$ so that the optimal value $Z^*$ satisfies $167 = \underline{Z} \leq Z^* \leq \bar{Z} = 190$.

**Remarks.** (i) If $Z'(p^B) = \mathrm{PO}(p^B)$, then $p^B$ is the optimal permutation.

(ii) All pair assignments with a cost in $B^*$ greater than $\Delta = \bar{Z} - \underline{Z}$ may be prohibited:

$$\text{if } B^*_{i,k} \geq \Delta \quad \text{then } B^*_{i,k} = +\infty.$$

For example, if we choose to assign $F_1$ to $L_1$, we obtain an assignment with a cost greater than or equal to $167 + 29 = 196$. $(Z(p) \geq B^*(p) + 167.)$

(iii) A reduction is all the more powerful since it produces a higher lower bound. We use RED2, which tries to decrease as much as possible the greatest element of the current matrix in order to minimize the importance of the quadratic term $Z'(p)$ in the objective function.

RED2 for $k = 2, \ldots, 2n$

find the greatest $m_{ij}$, $m_{i'j'}$,

if $\min_{j \neq i'} m_{i'j} > \min_{i \neq i'} m_{ij'}$, then $\alpha_{i'} = \min_{j \neq j'} m_{i'j}$, $m_{i'j} = m_{i'j} - \alpha_{i'}$,

else $\beta_{j'} = \min_{i \neq i'} m_{ij'} = m_{ij'} - \beta_{j'}$.

## 3. Branch and bound procedure

*A 'polytomic' branching scheme*

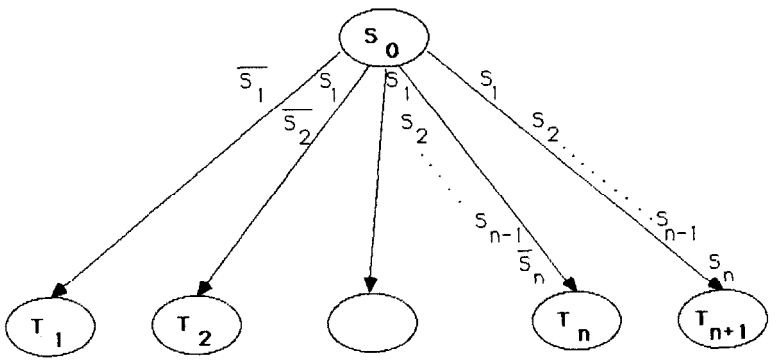
To simplify, we define the separation of the root node $S_0$. The associated matrix $B$, called $B^0$, gives us a lower bound. Let $P^{B^0}$ be the permutation that minimizes

$B^0(p)$:

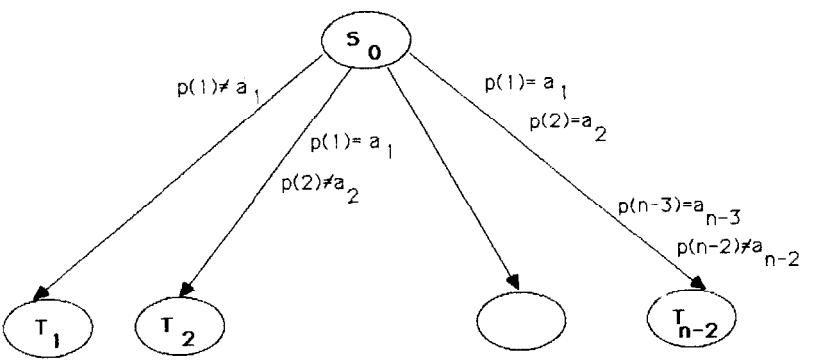
$$P^{B^0} = \begin{pmatrix} 1 & 2 & \cdots & n \\ a_1 & a_2 & \cdots & a_n \end{pmatrix}$$

For $i = 1, \ldots, n$, $S_i$ denotes the set of assignments which have the property $P_i$: $i$ is assigned to $a_i$. $\overline{S_i}$ denotes the complement of $S_i$ in $S_0$.

A separation of $S_0$ will be a partition of $(n+1)$ elements where: $T_1 = \overline{S_1}$, $T_2 = \overline{S_2} \cap S_1$, $T_n = \overline{S_n} \cap S_{n-1} \cap \cdots \cap S_1$, $T_{n+1} = S_n \cap S_{n-1} \cap \cdots \cap S_1$; $\bigcup_{i=1}^{n+1} T_i = S_0$ and $T_i \subset S_i$.




In fact, only the successor nodes $T_1, \ldots, T_{n-2}$ will be generated; the nodes $T_{n-1}$, $T_n$, $T_{n+1}$ are terminal nodes. $T_{n+1}$ corresponds to the feasible solution $p^{B^0}$, $T_n$ to an infeasible solution (if $p(1) = a_1, \ldots, p(n-1) = a_{n-1}$ then $p(a) = a_n$) and there is only one feasible solution $(p(1) = a_1, \ldots, p(n-1) = a_n, p(n) = a_{n-1})$ belonging to $T_{n-1}$. So, it is not necessary to create them.



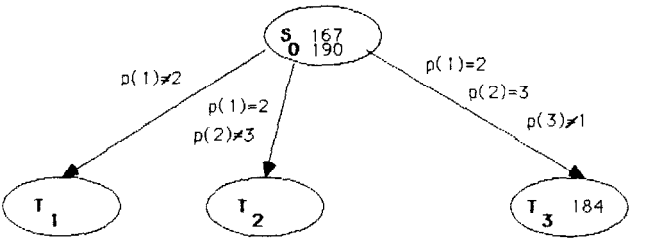

The choice of the first property $(p(1) = a_1)$ is based on the well known heuristic 'regret' or alternate cost:

$$\min_{j \neq a_1} b_{1j} + \min_{i \neq 1} b_{ia_1}$$

The pair assignments are ranked in descending order according to their alternate cost.

In the example, $p^{B^0} = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 3 & 1 & 4 \end{pmatrix}$; $r_{12} = 18$, $r_{23} = 13$, $r_{31} = 0$, $r_{41} = 0$.

Hence the first level of the tree will be:




$T_3$ is not created. But, the cost of $\begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 3 & 1 & 4 \end{pmatrix}$ is computed: 184.

*Bound computation*

At each node $T_k$ of the tree, we know the set of pair assignments included and the set of prohibited pair assignments. This partial solution of the assignment problem is represented by the partial permutation $\alpha$ defined on $I_k \subset N$ and we call $\beta$ the permutation defined on $\bar{I}_k$, complement of $I_k$; $\beta$ corresponds to a completion of the partial solution. First, we compute $B^{(k)}(\beta)$, i.e., for $i \in \bar{I}_k$,

$$b_{i\beta(i)} = k_{i\beta(i)} + 2 \sum_{j \in I_k} f'_{ij} d'_{\beta(i)\alpha(j)} + \mathrm{PO}(f'_{i\cdot}, d'_{\beta(i)\cdot}).$$

We solve the linear assignment problem with cost matrix $B^{(k)}$ to obtain $\underset{\sim}{Z}^{(k)}$. Let $p^{B(k)}$ be the optimal permutation. Then $\underset{\sim}{Z}^{(k)} = K(\alpha) + Z'(\alpha) + B(p^{B(k)}) - g$. Each pair assignment with a cost greater than $\Delta = Z^\circ - \underset{\sim}{Z}^{(k)}$ is prohibited, if $Z^\circ$ is the cost of the best known solution.

## 4. Results

The results presented here concern problems whose size is less then or equal to 12. Beyond size $n \geq 15$, the number of nodes in the state space tree required to store exceeds the system capacity ($30\,000 \times n - 1$). The problems we have tested are those of Nugent, Vollman and Ruml [14] and some example of our own (generated by a random number generator with uniform distribution in the interval [1, 99]).

The list of nodes are ranked in increasing order following their lower bound. We have tested two strategies: the first strategy ('best lower bound first') selects a node at the top, the other one ('deepest parent node first') at the bottom of the list. Due to the structure of the branching scheme, the second strategy is very interesting. The search tends to be longer but the greatest number of nodes in the list is smaller. This is very promising since the overflowing storage is the main cause for a program to stop before reaching the optimal solution for problem size $n \geq 15$.

Table 1. Running times on Cray 1 for small problem instances.

| Problems Nugent | Total number of nodes $N$ generated | Maximal number of nodes in the list $N$ | $N_{max}/N$ | Time $T$ (sec.) |
|---|---|---|---|---|
| size 5 | 16 | 9 | 56% | 0.03 |
| 6 | 69 | 19 | 27% | 0.19 |
| 7 | 134 | 37 | 27% | 0.56 |
| 8 | 428 | 281 | 65% | 2.9 |

$N$ = number of nodes generated during the search (terminal nodes excepted).
$N_{max}$ = maximal number of nodes in the list during the search.
$T$ = time to find the optimal solution and to verify optimality on a Cray 1 computer (automatic vectorization).

## 5. A parallel algorithm

We proposed general methods for designing distributed B&B algorithms well suited for asynchronous MIMD machines or computer networks in [9] and in particular, an algorithm [10] dedicated to a system with a modest number of processors and an efficient message passing scheme.

We now give an adaptation of these ideas to such a machine: a shared memory multiprocessors computer.

### *5.1. Main characteristics*

Our method is based upon the use of $N$ processes which are to be dealt with concurrently (where $N$ depends either on the number of physical devices such as processors, or on the size of the problem to be solved).

The distribution of the work among the different processes is done by giving access to a shared list which contains information about every node which is to be expanded. We associate a priority to each node; the priority has to be adapted to the problem to be solved. For a best-first strategy, the nodes with the least lower bound are selected first.

Hence, every active process finds, at the top of the list, the node it is going to expand.

Insertion of items is done whenever the expansion of a node generates several successors whose evaluation is less than the best known upper bound (feasible solution); these successors are then inserted in the list.

Table 2. Running times on Cray 1 for problem with size $n = 10$.

| Problems | $N$ | $N_{max}$ | $N_{max}/N$ | time $T$ |
|---|---|---|---|---|
| size 10 | | | | |
| strategy top 1 | 3586 | 2489 | 69% | 51 |
| bottom 2 | 4600 | 6 | 13% | 59 |

Table 3. Running times on Cray 1 for problem with size $n > 10$.

| Size | Strategy | Limited time | Best solution obtained | Distance $d$ at most | Number of nodes generated |
|---|---|---|---|---|---|
| 12 | 1 | 600 | 248982 | 7% | 22200 |
| Nugent 15 | 1 | 720 | 1172 | 11% | 12000 |

$d$ = distance from optimal solution at most $Z^\circ - \underline{Z}/Z^\circ$, $Z^\circ$ value of the best solution obtained, $\underline{Z}$ the last best lower bound when the program stops.

The best known upper bound (BKUB) is a shared variable which is updated whenever a local upper bound (lub), less than BKUB, is found at a node to be expanded.

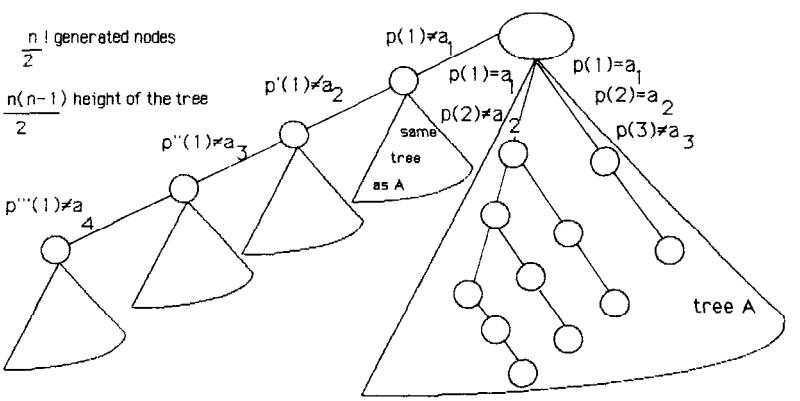
Items are suppressed either at the beginning or at the end of the list. The former case occurs whenever an inactive process looks for a new job, the latter case occurs whenever a lub is less than BKUB: every node whose evaluation is greater than lub is eliminated because it cannot lead to an optimal solution.

The algorithm terminates whenever the list is empty and all the processes are inactive.

### *5.2. Data structures*

The major data structures are matrices $F$ and $D$ ($n \times n$ square matrices where $n$ is the size of the assignment) and the list of nodes of the tree. This list is implemented as a heap (binary tree) in order to use fast existing algorithms to insert, sort and remove items. A polytomic branching scheme allows us to insert in the list several nodes at the same time.

The information that we keep in the list for a node is its lower bound, and what we call the context of a node. For an assignment problem, we need only the set of included pair assignments and the set of excluded ones. In order to limit the memory storage, we modify the branching rule. At the first separation, pair-assignments are ranked in decreasing order according to their alternate costs. This order will always be followed in the other separations.



### 5.3. *Definition of variables*

Variables used in multitask code can be categorized as follows according to the way they are allocated by the tasks that have access to them:

(a) Shared variables

LIST: list of items implemented as a heap; each item contains the evaluation of a node, all the pair-assignments included, and the number of prohibited pair-assignments; size $(\cdots, n-1)$,

NBLIST: number of nodes in LIST,

BKUB: best already known assignment (upper bound),

COUNT: number of active processes.

(b) Event variables (We use events for synchronization; a process can wait for events or post events that others may be waiting for.)

INSER: this event is sent when a process inserts items in the list that was empty.

(c) Variables local to one process

lub: upper bound (an assignment) associated with a node of the tree,

llb: lower bound associated with a node of the tree.

The critical region (segment of the code that has access to a shared resource) must be monitored because the program modules containing them have to run in parallel. This monitoring can be done by having one code segment set a lock when it enters a critical region (this amounts to the process setting up a flag to indicate that the shared data is being used). All other processes that run in parallel check the lock before they enter a corresponding critical region. The synchronization primitives are:

LOCK(−) and UNLOCK(−).

### 5.4. *The program*

**'Initial' process:**
**begin**
COUNT = 0; NBLIST = 0
LIST = root of the tree; FIN = false
Branch to this node and create its successors
Compute its upper bound (lub)
Compute lower bound (llb) for each successor of the root
BKUB = lub
**For** each successor node **do**
**if** llb < BKUB **then** insert this node in LIST
NBLIST = NBLIST + 1
Start concurrently several node processes
**end**

```
Node process:
begin
  end = false
  while end = false do
    lock (NBLIST)
    if NBLIST = 0 then
                    if COUNT = 0 then
                                    end = true
                                    unlock (NBLIST)
                                    send event (INSER)
                                  else
                                    unlock (NBLIST)
                                    wait event (INSER)
                  else
                    COUNT = COUNT + 1
                    select the node at the top of LIST
                    suppress top of LIST
                    NBLIST = NBLIST - 1
                    unlock (NBLIST)
                    create the successors of this selected node
                    compute upper bound (lub)
                    compute lower bound (llb) for each successor node
                    read BKUB
                    if lub < BKUB then
                                    lock (NBLIST)
                                    suppress in LIST each node with
                                      llb ≥ BKUB; decrease NBLIST
                                    BKUB = lub
                                    unlock (NBLIST)
                    lock (NBLIST)
                    for each successor node do
                      if lub < BKUB then
                        if NBLIST = 0 then send (INSER)
                        insert this successor in LIST
                        NBLIST = NBLIST + 1
                    COUNT = COUNT - 1
                    unlock(NBLIST
end
```

### *5.5. Experimental results*

We tried out our parallel algorithm on a Cray X-MP (a shared memory machine with four vector processors and 8M words of memory). The algorithm has been cod-

ed in FORTRAN 77 and the multitasking library of Cray has been used. First results have been obtained on an emulator called CREM running on a Multics machine BULL DPS 68, (more details in [11]) at INRIA, France and last results on the Cray X-MP 48 at CRAY RESEARCH, Minneapolis, USA (dedicated time).

We did two kinds of tests:

- in the first series the number of processes is increased from 1 to 3 on CREM, from 1 to 4 on Cray X-MP in order to see the improvement;
- the second series compares various search strategies in the B&B algorithm for a given number of processes.

### 5.5.1. *Speed-up*

We solve examples from Nugent using a *best first strategy* (Table 4). Speed-up here is computed as: $T_1/T_i$ where $T_1$ is the solution time for one process and $T_i$ the solution time for $i$ processes.

Table 4. Speed-up for small problems from Nugent on CREM

| Size $n$ | Example from | Number of processes | Simulated time (sec.) | Speed-up |
|---|---|---|---|---|
| | | 1 | 2.85 | 1 |
| 5 | Nugent | 2 | 3.18 | 0.90 |
| | | 3 | 3.31 | 0.86 |
| | | 1 | 9.71 | 1 |
| 6 | Nugent | 2 | 6.54 | 1.48 |
| | | 3 | 5.56 | 1.75 |
| | | 1 | 25.8 | 1 |
| 7 | Nugent | 2 | 14.85 | 1.74 |
| | | 3 | 11.27 | 2.29 |
| | | 1 | 126.50 | 1 |
| 8 | Nugent | 2 | 65.32 | 1.94 |
| | | 3 | 45.73 | 2.77 |

Of course, the execution times vary with regard to the distribution of nodes among the processes (Table 5).

Table 5. Example 'Nugent, size $n=7$', running times on CREM.

| Process 1 | | Process 2 | | Process 3 | | Total number of | Time |
|---|---|---|---|---|---|---|---|
| *NS* | *T* | *NS* | *T* | *NS* | *T* | nodes generated | (sec.) |
| 50 | 11.27 | 41 | 9.13 | 43 | 9.31 | 134 | 11.27 |
| 49 | 11.35 | 42 | 9.40 | 43 | 9.18 | 134 | 11.35 |
| 31 | 8.30 | 40 | 8.91 | 63 | 12.26 | 134 | 22.26 |
| 65 | 13.06 | 33 | 8.72 | 36 | 8.97 | 134 | 13.06 |

*NS* = number of nodes, *T* = time in seconds.

Speed-ups are not significant for small size instances ($n = 5$ or 6). On the Cray X-MP, we generated problems with $n = 10$ and coefficients from a uniform distribution in range [1, 99]. The average speed-up is given in Table 6.

Table 6. Running time on Cray X-MP.

| $n = 10$ | Number of processes | Average Speed-up | Average time |
|---|---|---|---|
| Best | 1 | | 30.74 |
| first | 2 | 1.97 | 15.57 |
| strategy | 3 | 2.95 | 10.43 |
| | 4 | 3.90 | 7.89 |

For problem size $n = 10$, the example called Nugent 12 has been solved in 5 mn 12s with four processes (Table 7).

Table 7. Running time for 'Nugent $n = 12$', on Cray-X-MP.

| $T_1$ | $N_1$ | $T_2$ | $N_2$ | $T_3$ | $N_3$ | $T_4$ | $N_4$ |
|---|---|---|---|---|---|---|---|
| 308 | 20 831 | 301 | 20 833 | 312 | 20 843 | 298 | 20 872 |

$T_i$ = time in seconds on processor $i$; $N_i$ = number of nodes on processor $i$.

For problems with $n \geq 10$, the improvement in using $n$ processors is very close to $n$.

### 5.5.2. *Testing of search strategies*

For sequential B&B algorithms it has been shown [6] that a best first strategy is optimal in the sense that it minimizes the total number of nodes generated under the following conditions: no ties occur among the lower bounds, branching and bounding depend only on the direct history of the tree (i.e. the path from the root to the node under examination).

Hwang Lai and Sahni [17] and also Burton et al. [3] prove that with a best first strategy a parallel B&B algorithm may require more or less time than the corresponding sequential algorithm by an arbitrary factor; in their proof they assume that all

Table 8. Running time for small problems on CREM.

| | Strategy 1 | | | Strategy 2 | | | Strategy 3 | | |
|---|---|---|---|---|---|---|---|---|---|
| | $N_m$ | $N$ | $T$ | $N_m$ | $N$ | $T$ | $N_m$ | $N$ | $T$ |
| Mug7 | 36 | 134 | 11.27 | 7 | 174 | 11.49 | 23 | 185 | 11.39 |
| Mat7 | 18 | 58 | 6.69 | 4 | 69 | 9.06 | 8 | 69 | 7.21 |
| Nug8 | 282 | 428 | 45.73 | 9 | 443 | 45.25 | 69 | 481 | 40.85 |
| Mat8 | 66 | 380 | 40.62 | 7 | 386 | 41.19 | 39 | 380 | 33.40 |

$N_m$ = maximal number of nodes stored together in the list.
$N$ = total number of nodes generated.
$T$ = simulated time in seconds.

processors are synchronized (which is not our case) and that the priority function is regular (no ties).

So, we studied on CREM the influence of a given strategy with a fixed number of processors (e.g. 3). These strategies are those defined above (2.4): (1) best first, (2) deepest parent node first, (3) random.

Strategy 1 minimizes the number of nodes, but with strategies 2 and 3, the examination of a node (branching and bounding) is faster.

On Cray X-MP, Strategies 1 and 2 for the problem with $n = 10$ and range [1, 99] are comparable.

Table 9. Running times on Cray-X-MP for problem with size $n = 10$.

| Number of processes | Strategy 1 average time | speed-up | Strategy 2 average time | speed-up | Strategy 3 average time | speed-up |
|---|---|---|---|---|---|---|
| 1 | 30.74 | | 31.10 | | 47.38 | |
| 2 | 15.57 | 1.97 | 15.62 | 1.99 | 24.17 | 1.96 |
| 3 | 10.43 | 2.95 | 10.55 | 2.95 | 15.60 | 3.04 |
| 4 | 7.89 | 3.90 | 7.93 | 3.92 | 11.91 | 3.98 |

## Conclusion

We gave a parallel implementation of a QAP problem on a multiprocessor machine with shared memory (CRAY-XMP).

For the examples we considered (size 10) we observed that the speed-up is nearly equal to the number of processors (up to 4 processors). This proves that an adequate parallelization of the QAP has been found.

Moreover, these results also show that asynchronous parallel machines are better suited to the implementation of parallel B&B algorithms than synchronous ones [9].

From the point of view of memory occupancy, it is also interesting to notice that in some problem instances, the size of the search tree decreases when the number of processors increases.

More generally, we have shown that our strategies provided a good trade off between speed-up and memory size.

## Acknowledgements

The author is very indebted to Cray France and Cray Research Minneapolis, for supporting this research with computer time and for helpful discussions.

## References

[1] N.N. de Abreu and P.O. Boaventura, An algebraic and combinatorial study of the quadratic assignment problem, Euro VII, Bologne, Italy, June 1985.

[2] R.E. Burkard, Some recent advances in quadratic assignment problems, Math. Programming, (1984) 53–68.

[3] F.W. Burton, M.M. Huntbach, G.P. McKeown and V.J. Rayward Smith, Parallelism in branch and bound algorithms, Mathematical Algorithms group-2, Internal report CSA/3, University of East Anglia, Norwick, 1983.

[4] O.I. El Dessouki and W.H. Huen, Distributed enumeration on network computers, IEEE Trans. Comput. 29 (1980) 818–825.

[5] G. Finke, R.E. Burkard and F. Rendl, Quadratic assignment problems, School on Combinatorial Optimization, Rio de Janeiro, Brazil, July 1985.

[6] B. Fox, J. Lenstra, A. Rinnooy Kan and L. Schrage, Branching from the largest upper bound folklore and fact, Europ. J. Oper. Res. (1978) 191–194.

[7] P.C. Gilmore, Optimal and suboptimal algorithms for the quadratic assignment problem, J. SIAM 10 (1962) 305–313.

[8] G.A.P. Kindervater, Parallel enumerative methods, EURO VII, Bologna, Italy, June 1985.
G.A.P. Kindervater and H.W.J.M. Trienekens, Experiments with parallel algorithms for combinatorial problems, Report OS-R8512, Centrum voor Wiskunde en Informatica, Amsterdam, Nov. 1985.

[9] I. Lavallée and C. Roucairol, A parallel branch and bound algorithm for asynchronous MIMD machines and computer networks, Rapport interne CNRS GR22, Paris VI, avril 1983.

[10] I. Lavallée and C. Roucairol, Parallel branch and bound algorithms, EURO VII Congress, Rapport interne MASI, Université Paris 6, 1985.

[11] P. Laurent and C. Verleye, Procedures arborescentes sur machines parallèles et réseaux de processeurs, Mémoire d'ingénieur IIE-CNAM, Evry, 1985.

[12] T. Mautor and D. Savoye, Etudes d'heuristiques pour le problème d'affectation quadratique, Mémoire d'ingénieur CNAM-IIE, 1982.

[13] J. Mohan, A study in parallel computation: the traveling salesman problem, Carnegie Mellon University, CMU-CS-82-136, 1982.

[14] C.E. Nugent, T.E. Vollmann and J. Ruml, An experimental comparison of techniques for the assignment of facilities to locations, J. Oper. Res. 16 (1969) 150–173.

[15] C.D. Roucairol, A reduction method for quadratic assignment problem, Operations Research Verfahren/Methods of Operations Research 32 (1979) 185–187.

[16] C.D. Roucairol, An efficient branching scheme in branch and bound procedures, Tims XXVI, Paper no. 42, Masi, Université Paris 6, Paris, 1984.

[17] S. Sahni and Ted Hwang Lai, Anomalies in branch and bound, Comm. ACM 27 (6) (1984) 594–602.

[18] B.W. Wah and Y.W. Ma, Manip – a multicomputer architecture for solving combinatorial problem extremum search problems, IEEE Trans. Comput. (5) (1984) 377–389.